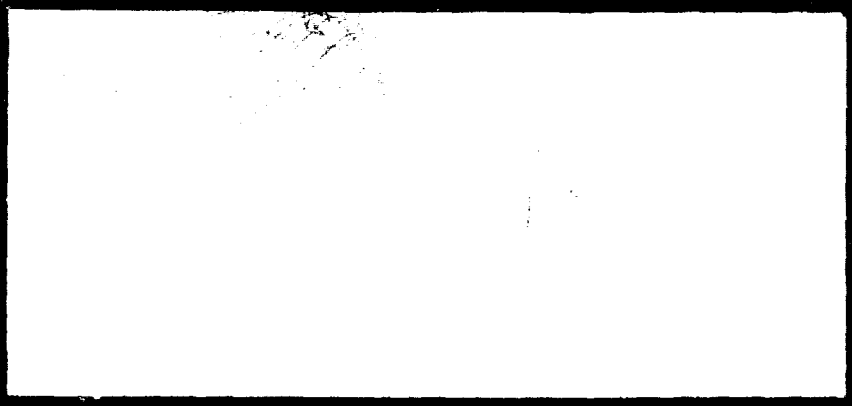

A STUDY OF METHODS TO PREDICT AND MEASURE THE TRANSMISSION OF SOUND THROUGH THE WALLS OF LIGHT AIRCRAFT

Research Contract #0226-51-1288

INTEGRATION OF CERTAIN SINGULAR BOUNDARY ELEMENT INTEGRALS FOR APPLICATIONS IN LINEAR ACOUSTICS

Sponsored by

NASA
Hampton, VA 22365

Report No. 0226-16 HL 85-21

Submitted by:

Dan Zimmerle, Graduate Research Assistant
Robert J. Bernhard, Principal Investigator

Approved by:

Raymond Cohen, Director
Ray W. Herrick Laboratories

July 1985

CONTENTS

1. INTRODUCTION

It is well known that boundary element techniques require the integration of a characteristic solution over elements which define the boundary of the domain. The characteristic solution is usually a function of the distance from a "root" node to the surface of the element over which the integral is being computed. When the "root" node is outside of the element, the integrals may be performed in a straight-forward fashion using standard numerical integration techniques like Gaussian quadrature. However, when the node is in the element, the integral contains an integrable singularity which will not integrate accurately using Gaussian quadrature. This report will discuss an alternative method for performing this integral.

The method proposed separates the integral of the characteristic solution into a singular and non-singular part. The singular portion is integrated with a combination of analytic and numerical techniques while the non-singular portion is integrated with standard Gaussian quadrature. The method may be generalized to many types of sub-parametric elements.

This report will consider only the integrals over elements containing the root node, and will deal only with the characteristic solution for linear acoustic problems. However, the method described may be generalized to most characteristic solutions.

2. BEM INTEGRALS OVER AN ELEMENT CONTAINING THE ROOT NODE

Suppose the root node is node n ($1 < n < m$). To denote distance from node n, write r as r_n.

$$I_i = \int_{A_e} N_i \frac{e^{-jkr_n}}{r_n} dA \qquad i = 1,2,\ldots,m \qquad (1)$$

where

I_i - contribution (i.e., integral) to coefficient of node i.

N_i - shape function for node i

A_e - area of element e (region of integration)

r_n - distance from root node to any point in the element

k - constant from Helmholtz equation

m - number of nodes in element e

Several significant properties of the shape functions are:

1) All $\{N_i : i \neq n\} \rightarrow 0$ as $r_n \rightarrow 0$

2) $N_n \rightarrow 1$ as $r_n \rightarrow 0$

3) $N_n = 1 - \sum_{\substack{i=1 \\ i \neq n}}^{m} N_i$

From these observations we note that the integrand of I_i is singular only when i=n. For all other integrals, $N_i \rightarrow 0$ as $r_n \rightarrow 0$ quickly enough to avoid singularity. Thus, I_i, for $i \neq n$

may be integrated numerically without undo difficulty.

For i=n an integrable singularity exists. First, rewrite the integral I_n as:

$$I_n = \int_{A_e} N_n \frac{e^{-jkr_n}}{r_n} dA$$

$$= \int_{A_e} \frac{e^{-jkr_n}}{r_n} dA - \sum_{\substack{i=1 \\ i\neq n}}^{m} I_i \qquad (2)$$

We may therefore break I_n into a singular part,

$$I_o = \int_{A_e} \frac{e^{-jkr_n}}{r_n} dA \qquad (3)$$

and a non-singular sum of previously calculated integrals,

$$I_n = I_o - \sum_{\substack{i=1 \\ i\neq n}}^{m} I_i \qquad (4)$$

The problem, then, is reduced to calculating I_o.

2.1 Computation of I_o

Note:

$$I_o = \int_{A_e} \frac{\cos kr_n}{r_n} dA + j \int_{A_e} \frac{\sin kr_n}{r_n} dA \qquad (5)$$

but

$$\lim_{r_n \to 0} \frac{\sin kr_n}{r_n} = k$$

Therefore, the second integral may be evaluated numerically, if desired. Temporarily both will be kept together.

2.2 Linear Elements

For linear elements sides are straight lines:

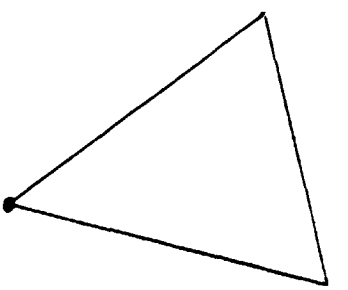

Figure 1. Type T-3 Element

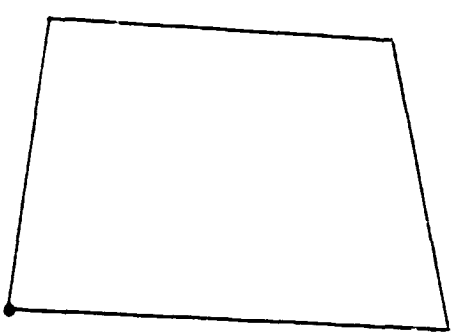

Figure 2. Type Q-4 Element

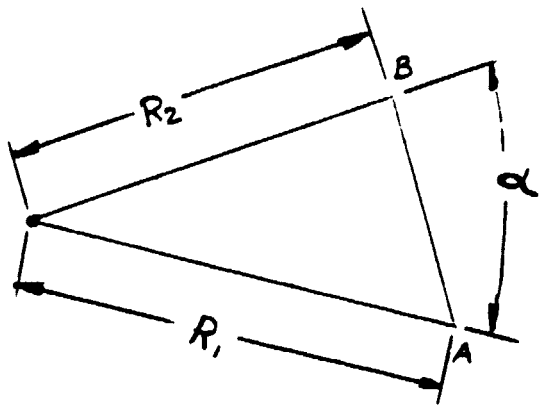

Figure 3. Polar Coordinate Definition for T-3 Element

Rephrasing I_o in local polar coordinates:

$$I_o = \int_{A_e} \frac{e^{-jkr_n}}{r_n} r_n dr_n d\theta = \int_0^{\alpha} \int_0^{R_1(\theta)} e^{-jkr_n} dr_n d\theta \qquad (6)$$

where R_1 is the equation of line AB in polar coordinates.

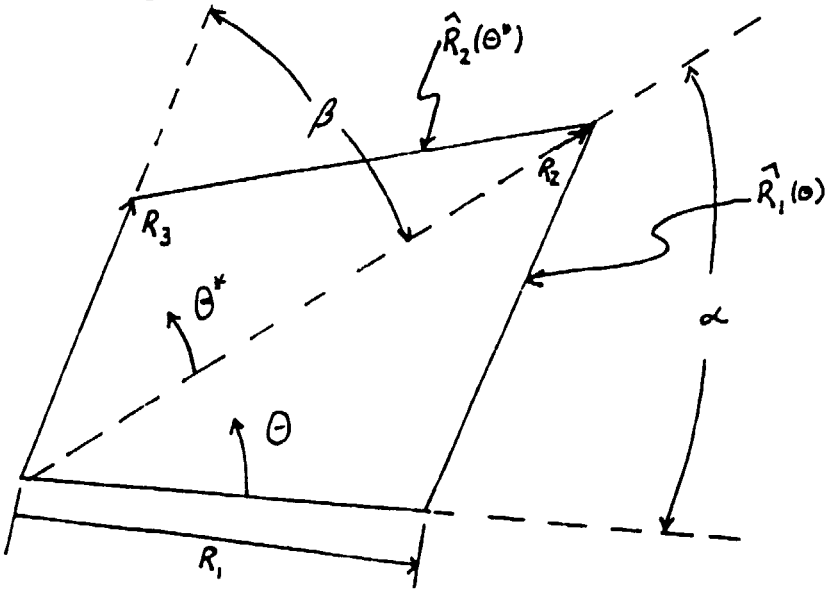

Figure 4. Polar Coordinate Definition for Q-4 Element

For a quadrilateral element

$$I_o = \int_0^{\alpha} \int_0^{R_1(\theta)} e^{-jkr_n} dr_n d\theta + \int_0^{\beta} \int_0^{R_2(\theta^*)} e^{-jkr_n} dr_n d\theta^* \qquad (7)$$

Thus, an expression for $R_1(\theta)$ and $R_2(\theta^*)$ must be found

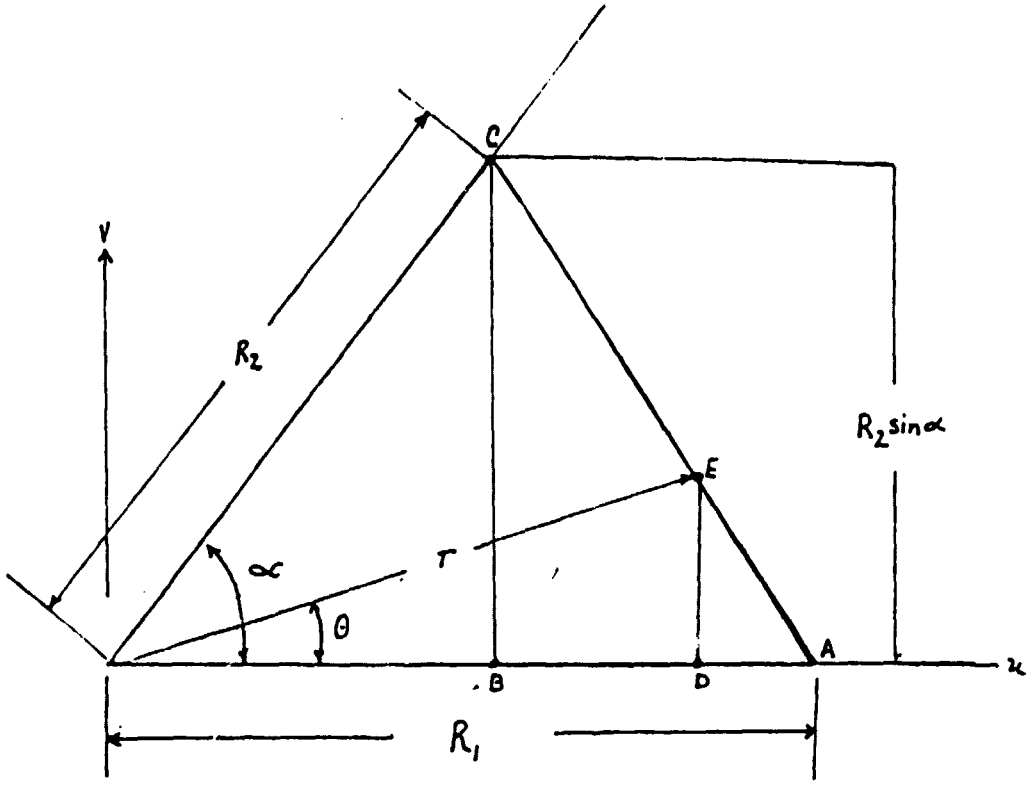

Figure 5. Definition of $R(\theta)$

As shown in Figure 5, by similar triangles ABC ADE

$$\frac{R_2 \sin\alpha}{r\sin\theta} = \frac{R_2\cos\alpha - R_1}{r\cos\theta - R_1}$$

$$(r\cos\theta - R_1)(R_2\sin\alpha) = (r\sin\theta)(R_2\cos\alpha - R_1)$$

$$r(R_2\sin\alpha\cos\theta - R_2\cos\alpha\sin\theta + R_1\sin\theta) = R_1R_2\sin\alpha$$

but,

$$R_2(\sin\alpha\cos\theta - \cos\alpha\sin\theta) = R_2\sin(\alpha - \theta)$$

$$R = r = \frac{R_1R_2\sin\alpha}{R_2(\sin\alpha\cos\theta - \cos\alpha\sin\theta) + R_1\sin\theta}$$

or

$$R = \frac{R_1 R_2 \sin\alpha}{R_2 \sin(\alpha - \theta) + R_1 \sin\theta}$$

Therefore:

$$R_1(\theta) = \frac{R_1 R_2 \sin\alpha}{R_2 \sin(\alpha - \theta) + R_1 \sin\theta} \tag{8}$$

$$R_2(\theta^*) = \frac{R_2 R_3 \sin\beta}{R_3 \sin(\beta - \theta^*) + R_2 \sin\theta^*}$$

From Equation 6, the integral over r is non-singular and easily integrated. Equations 6 and 7 can now be partially evaluated numerically by

$$\int_0^\gamma \int_0^{R(\theta)} e^{-jkr_n} dr_n d\theta = \int_0^\gamma \left[\frac{e^{-jkr_n}}{-jk} \right]_0^{R(\theta)} d\theta$$

$$= \frac{j}{k} \int_0^\gamma \left[e^{-jkR(\theta)} - 1 \right] d\theta$$

$$= \frac{j}{k} \int_0^\gamma e^{-jkR(\theta)} d\theta - \frac{j\gamma}{k} \tag{9}$$

This integral is easily evaluated numerically using Gauss quadrature over θ.

3. EXTENSION TO HIGHER-ORDER ELEMENTS

For many applications, linear triangular and quadrilateral elements are sufficient. However, for some applications, higher-order (e.g., quadratic or cubic) elements may provide an increase in accuracy. There are two types of elements with higher interpolation orders that are of interest here -- iso-parametric and sub-parametric elements.

In sub-parametric elements the geometry of the element is defined by a lower order of interpolation than the function values. If the geometry is defined in terms of linear shape functions the method of determining I_o is exactly the same for all sub-parametric elements as the method presented above, where m is the number of nodes in the element.

In iso-parametric elements, however, the geometry is defined by higher-order interpolation, as shown in Figure 6.

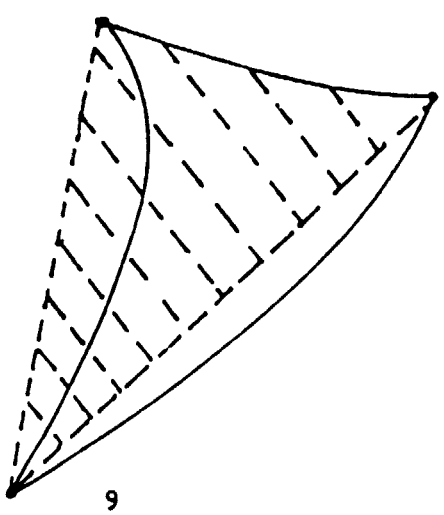

Figure 6. Isoparametric Element Displacements

Since the boundary of the element is no longer defined by straight lines, the techniques presented above will not integrate I_o correctly, as shown by the shaded area in the figure. In this author's experience computing I_o for an iso-parametric element is much more difficult than computing it for a sub-parametric element.

In many applications the geometry of the region is neither complex enough to warrant iso-parametric elements nor critical enough that it needs to be rspresented exactly. In addition, there is some evidence which indicates that higher-order elements do not improve the accuracy of the method significantly. For these reasons it was decided to concentrate on the sub-parametric techniques, which is much simpler and should give a good indication of the advantages, if any, of using higher-order elements. If necessary, iso-parametric capability may be developed at a later date.

3.1 Computation of the Integral for Sub-Parametric Elements

At this juncture it is interesting to look at methods for computing I_o in general -- i.e., covering the range of possible root nodes and element shapes.

The root node must be one of three types:

1. Corner node (all element types), as shown in Figure 7
2. On a side of the element (quadratic and cubic types), as shown in Figure 8

3. In the interior of the element (certain cubic types), as shown in Figure 9

In each case the root node is marked with a dot. All other nodes used to define the geometry of the elements are marked with an "x". Letters marking each of the sub-triangles are enclosed in circles.

Note that in each case the element may be divided into from one to four triangles by connecting the root node with each of the other corner nodes in the element. The integral for the entire element is simply the sum of the integral for each of the triangles. Also note the following: 1) One or two of the triangles will collapse to lines (i.e., have area of zero) for certain root nodes; 2) the number of subtriangles in each element is equal to the number of corner nodes in the element. By using the test "is the area of this triangle equal to zero" it is possible to write a general algorithm to compute I_o for any sub-parametric element as shown in Table I.

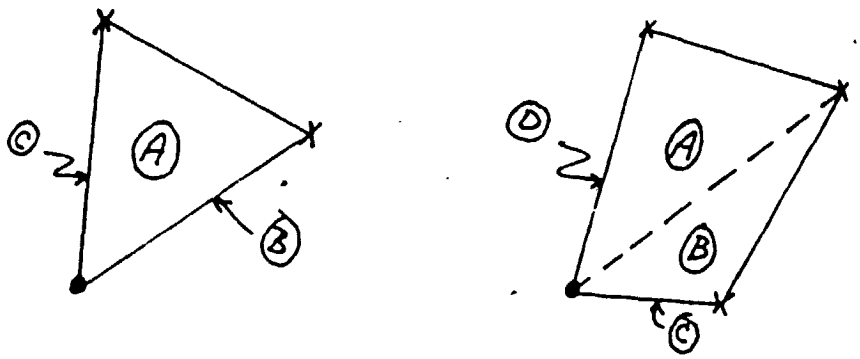

Figure 7. Root Node as a Corner Node

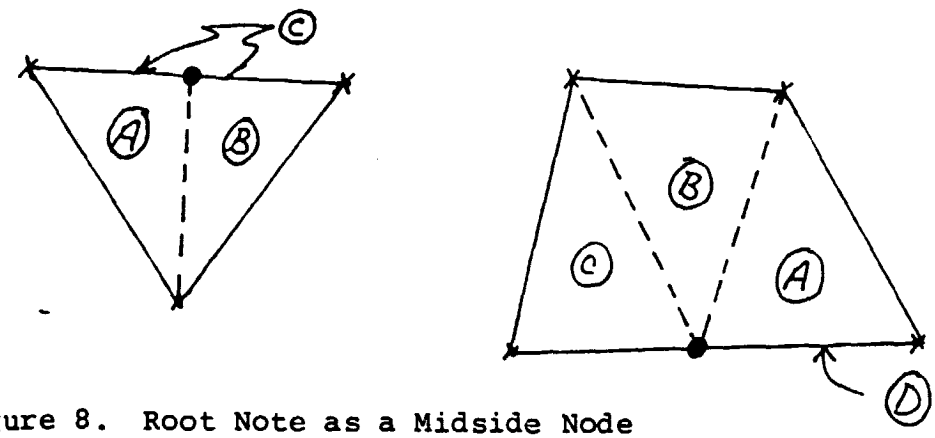

Figure 8. Root Note as a Midside Node

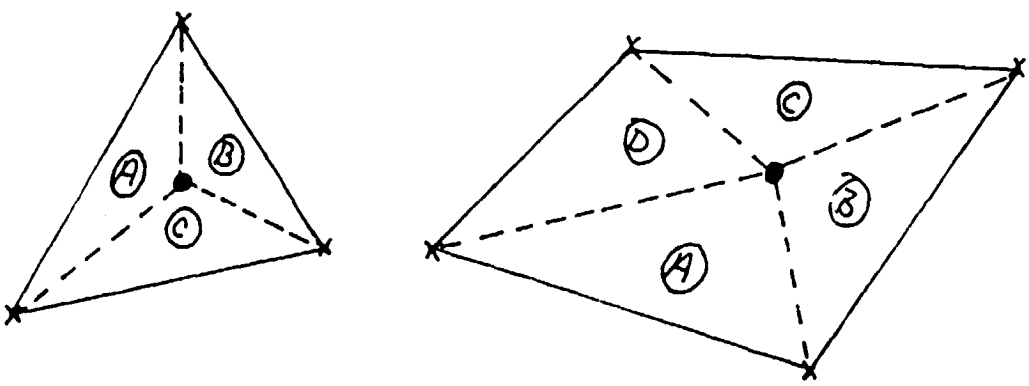

Figure 9. Root Node as an Interior Node

TABLE I.

6.1 Algorithm to Calculate I_o

A) Main Routine:

1) Set the sum I to zero

2) FOR i = 1 to number of corner nodes in the element (S) DO

   a) Select 2 corner nodes to create a triangle:

      a = i

      b = i + 1

      if (b > S) b = 1

   b) Calculate contribution to I for this triangle using the routine below:

   c) Add contribution to I.

3) END

B) Routine to calculate contribution to I for each triangle.

1) Calculate area of the triangle.

2) IF (the area of the triangle is zero) THEN

   a) return a value of zero
   ELSE

   b) compute the integral over the triangle (n,a,b) using Eq. 9.

3) END

4. INTEGRALS OF NORMAL DERIVATIVES

Up to this point, only integrals of the characteristic solution have been considered. However, it is well known that integrals of the normal derivative of the characteristic solution must also be computed.

If an element is a portion of a plane, the normal to the surface is perpendicular to the vector extending from the root node to any point on the element. This is true for all triangles formed with linear sides, and is true for quadrilaterals provided opposite sides lie in the same plane. In such cases where the normal and the "position" vector are perpendicular the normal derivative is zero.

Sub-parametric elements, which are being used exclusively in this report will be planar for all triangular elements. Sub-parametric quadrilateral elements may also be defined as planes with very little loss of versatility. Therefore, for sub-parametric elements, all integrals of the normal derivative are zero, and thus, very easy to compute.

5. CONSTRUCTION OF A BOUNDARY ELEMENT PROGRAM

In order to properly utilize the integration routines presented following this chapter, it is necessary to place them in the intended type of program structure. This section will briefly explain the type of program structure for which they were designed.

The major operations of the program are:

1. Input mesh definition into the program.
2. Assemble the system of equations.
3. Add internal sources to the equation system (optional).
4. Generate additional equations for overdetermined system.
5. Solve the system of equations.
6. Extract results.

Of course, this flow chart will vary somewhat if, for example, a line by line equation solver is used.

The most complex step of the program is the assembly of the system of equations. This step, is discussed in more detail in Table 2. Figure 10 is a flow-of-control diagram, showing which routines are called by, or connected to other routines.

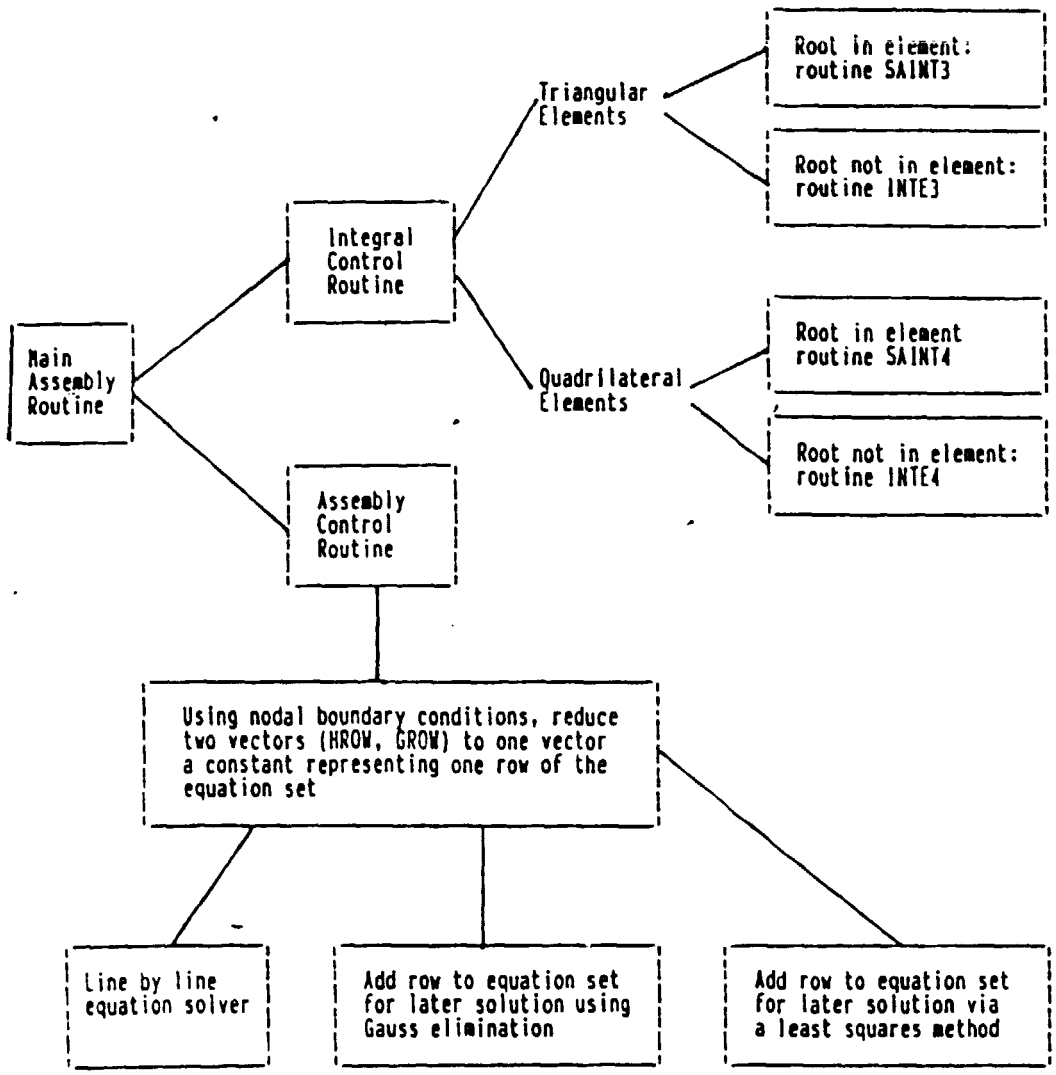

Figure 10. Flow of Control for Assembling the System of Equations.

TABLE II

Assembly of the System of Equations

0. Define complex vectors HROW, GROW of length NOD.

1. FOR i = 1 to the number of nodes (NOD) DO

    a. Set the vectors HROW, GROW to zero. These vectors will hold, respectively, the integrals of the normal derivative of the characteristic solution (H) and the characteristic solution (G), respectively.

    b. FOR j = 1 to the number of elements (NEL) DO

        i. Localize coordinates and node numbers for element j into temporary arrays.

        ii. IF j is triangular THEN

            1. IF node i is in element j THEN

                call integration routine for a triangle containing the root node (SAINT3).

            1, ELSE

                call integration routine for a triangle not containing the root node (INTE3).

        ii, ELSE

            1. IF node i is in element j THEN

                call integration routine for a quadrilateral containing the root node (SAINT4).

            1, ELSE

                call integration routine for a quadrilateral not containing the root node.

        iii. Add the integrals just computed to their proper locations in the vectors HROW, GROW.

    b, END of loop on the elements, j.

    c. Add the row of integrals for node i, accumulated in HROW and GROW to the coefficient matrix following the boundary conditions specified for each node.

1, END of loop on the nodes, i.

The tests in 1.b.ii of Table 2, which are used to select the type of integral to be computed, should properly be placed in a separate routine for easy maintenance. However, since the tests are quite simple and the integration routines very flexible, it should be more efficient to incorporate the tests directly into the main assembly routine. Step 1.c of Table 2 is dependent upon the type of solution procedure chosen, and should be placed in a separate routine. This subject will be discussed more later.

5.1 Construction of the Integration Routines

A listing of the integration routines (which is slightly out-of-date, but all that was available at the time of this writing) is given in the Appendices, along with shape function and certain other utility routines. This section will briefly discuss the structure of the integration routines.

The routines may be divided into three groups:

1. Computation of the I_o integral for an element
   COINTE, IOINTE

2. Computation of the integrals for an entire element
   SAINT3, INTE3, SAINT4, INTE4

3. Utility routines and shape functions SHAQUA, SHATRI, NORMAL along with the subroutines to SHAQUA:
   SH2DQQ, SH2DCQ

The routines in 1 and 3 are called by the routines in 2 and are

transparent to the remainder of the program.

5.1.1 The Routines to Compute I_o The routines COINTE and IOINTE are a direct implementation of the method described previously. COINTE divides the sub-parametric element into the number of sub-triangles. IOINTE is called by COINTE to calculate the integral over each sub-triangle. If one angle of the sub-triangle is 180 degrees (which is equivalent to asking "is the area zero", and is much easier to calculate), COINTE returns a value of zero.

5.1.2 The Routines to Compute the Element Integrals INTE3 and INTE4 compute the integrals for triangles (thus the 3) and quadrilaterals, respectively, for the cases when the root node is not in the element. These are straight forward implementations of standard Gaussian quadrature.

SAINT3 and SAINT4 are images of INTE3 and INTE4 except for two modifications. First, during the Gaussian quadrature, the integral for the root node is not calculated, since it is singular and inaccurate. Second, additional statements are added at the end of the routine to call COINTE to compute the I_o integral for the element. I_o is then used to compute the integrals for the root node.

The "SAINT" and "INTE" routines are not combined because a large number of tests would be needed within the routines to decide whether the root node is in the element. These tests would make the computation much less efficient.

Integration routines for triangular and quadrilateral elements are not combined in one routine because certain operations may be eliminated when triangular elements are handled separately. Also, the dimensions of the arrays and certain DO loop parameters may be specified as constants, which, on some compilers, is more efficient than specifying variables.

5.1.3 Utility Routines The utility routines contain NORMAL, which calculates the normal vector for a triangle with linear sides, and the shape function routines. The shape functions are simplifications of finite element shape functions (the derivatives of the shape functions need not be calculated) for 2-dimensional problems.

The shape functions have the capability to handle hierarchal nodes, as proposed by Zienkiewicz. Any of the midside nodes of the higher order elements may be removed, reducing the number of degrees of freedom and the order of the element. To remove node i, for example, specify the ith position in the element connectivity as zero, i.e., ID(i) = 0.

The integration routines assume that the quadrilateral elements are plane, or nearly plane. To minimize the error due to deviations from plane geometry, an "average" normal vector is used. However, results with non-plane elements may not be good.

Prior to any call to an integration routine, the Gaussian quadrature points must be specified in the common blocks QPOINT (quadrilateral elements), TPOINT (triangular elements) and SPOINT

(one dimensional Gauss points to integrate the θ integral for I_o).

5.2 Assembly of HROW, GROW into the Coefficient Matrix

Step 1.c in Table 2 for the assembly of the system of equations is in need of a few additional comments. As shown in the flow-of-control diagram, an additional routine is responsible for converting the integrals stored in HROW and GROW into a single equation for node i. The equation is then assembled into the system by the proper routine for the type of solution method used.

The conversion of the integrals into an equation is a simple operation based on the boundary conditions for the nodes. The program BOUN3D, originally written for three dimensional heat transfer contains all of the instructions necessary for this operation. The comments in the program should be self-explanatory.

Another pressing problem with boundary conditions is which boundary condition to enforce at a corner where different conditions are specified on each side of the corner. The system used in BOUN3D, and suggested for this program is to place two (or more) nodes close to the corner. The different boundary conditions may be specified on different nodes.

When the integrals are converted to an equation the correction factor for the included angle at the node must also be

incorporated. It is suggested the angle be calculated from the geometry of the mesh, rather than the integrals, as is usually done for heat transfer problems. This task is probably handled best within a mesh generation program. Note that an additional array will have to be added to BOUN3D to hold the angle at each node.

Finally, a note about equation solvers needs to be given. The usual manner of solving unsymmetric full systems by Gauss elimination has proven quite satisfactory in the past, and is suggested for this program. Iterative methods, such as SOR, BSOR and Gauss-Seidel usually will not converge at a reasonable rate.

A possible variation on Gauss elimination would take advantage of the fact that boundary element equations are formed completely, one at a time. Since the complete equation is available at once, it is possible to eliminate the first i-1 coefficients from row i at the time the row is added to the coefficient matrix. This "line by line" elimination would reduce the amount of storage needed for the matrix from N * N to N * (N -1)/2. However, it would make it impossible to pivot the matrix.

Finally, by separating the formation of the equation from its addition to the coefficient matrix, it will be easy to add specialized solution routines. For example, for some least squares methods, the number of equations and variables are not equal. Such solvers often require specialized assembly methods.

6. CONCLUSIONS

This work has provided several insights which may be useful.

1. Sub-parametric elements seem to be the most efficient means of utilizing higher-order elements. However, there is very little evidence to suggest that higher-order elements will greatly improve the accuracy of the solution.

2. Attention should be given to a method(s) of performing an iterative "design" type problem solution. It is very expensive to iterate using boundary element because the system of equations needs to be decomposed at each iteration. This could become a very pressing problem whenever iterations are necessary, such as a noise path identification application.

3. For problems where all of the nodes are given the same boundary conditions (such as an impedance) there should be very little problem imposing boundary conditions (i.e., the problems that haunt solid mechanics and heat transfer applications should not appear). This is an area that can be ignored until a serious problem crops up

4. The solution of the system of equations will continue to be the most time consuming part of the analysis. Some improvement must be made in this area, but such improvements probably not a relevant concern at this time.

APPENDICES

APPENDIX A: Subroutines COINTE and IOINTE

```
------------------------------------------------------------------
    function cointe(korn,iroot,x,y,z,ak)
------------------------------------------------------------------
 Computes the Io integral over the element by dividing the (sub-)
    parametrix element into a number of linear triangular elements
------------------------------------------------------------------
 Inputs:
     iroot - local number of the root node
       npe - number of nodes in the element (used to determine wheter
             element is triangular or quadrilateral)
     x,y,z - element nodal coordinates
        id - connectivity for element
        ak - constant from Helmholtx equation
------------------------------------------------------------------
 Outputs:
    cointe - value of the Io integral for the element
------------------------------------------------------------------
 Calls:
    iointe - computes Io for individual triangles in element
------------------------------------------------------------------
 Notes:
   1) iointe requires gauss integration points for one dimension.  These
      must be set before routine is called.
   2) Intended only for sub-parametric elements.  Serious errors may
      occur if used with parametric elements.
------------------------------------------------------------------

    implicit real*8 (a-h,o-z)
    complex*16 cointe,iointe
    dimension x(korn),y(korn),z(korn)
---------------------- arrays hold points for the nodes of subtriangle
    dimension xx(3),yy(3),zz(3)
---------------------- find element type, initilize variables
    cointe = (0.,0.)
    xx(1) = x(iroot)
    yy(1) = y(iroot)
    zz(1) = z(iroot)
---------------------- divide element into subtriangles and integrate each
    il = korn
    do 100 i = 1,korn
       i2 = i
       if(il .eq. iroot .or. i2 .eq. iroot) goto 100
          xx(2) = x(il)
          xx(3) = x(i2)
          yy(2) = y(il)
          yy(3) = y(i2)
          zz(2) = z(il)
          zz(3) = z(i2)
          cointe = cointe + iointe(xx,yy,zz,ak)
DO        il = i2
```

```
      return
      end
-------------------------------------------------
      function iointe(x,y,z,ak)
-------------------------------------------------------------------------
  Computes the constant integral over a triangular element or a triangular
     portion of an element
-------------------------------------------------------------------------
  Inputs:
   x,y,z - the coordinates of the triangle.  The root node is the first in
           the arrays.
      ak - constant from the helmholtz equation
-------------------------------------------------
  Outputs:
   iointe - value of the integral of:
              exp(-jkr)/r dA
-------------------------------------------------------------------------
  Notes:
    1) all reals are double precision
    2) the integral is performed using both analytic and numerical techniques
    3) if the area of the triangle is zero, iointe = 0
-------------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(3),Y(3),Z(3)
      complex*16 iointe
------------------- GAUSS POINTS FOR NUMERICAL INTEGRATION
      COMMON /SPOINT/ ETA(16),W(16),NINPT
---------------- TOP IS THE DOT PRODUCT OF R1 AND R2
      TOP =  (X(2) - X(1)) * (X(3) - X(1))
     *     + (Y(2) - Y(1)) * (Y(3) - Y(1))
     *     + (Z(2) - Z(1)) * (Z(3) - Z(1))
      R1 = DSQRT((X(2)-X(1))**2+(Y(2)-Y(1))**2+(Z(2)-Z(1))**2)
      R2 = DSQRT((X(3)-X(1))**2+(Y(3)-Y(1))**2+(Z(3)-Z(1))**2)
---------------- DOT R1 & R2 TO FIND ANGLE OF ELEMENT
      COSA = TOP/(R1* R2)
---------------- if cosa = 1, the triangle is collapsed to a line
      if(dabs(dabs(cosa)-1.d0) .le. 1.d-4) then
         iointe = (0.d0,0.d0)
         return
      end if
---------------- compute the Io integral for the triangle
      ALPHA  = DACOS(COSA)
      SINA   = DSIN(ALPHA)
---------------- NUMERATOR OF R-CARAT
      TOP    = R1 * R2 * SINA
      A2     = ALPHA * 0.5D0
      iointe = (0.d0,0.d0)
----------------- INTEGRATE THE THETA INTEGRAL
      do 100 i=1,ninpt
----------------- RC IS r-carat
      rc = top/(R2*dsin(a2*(1.d0-eta(i)))+R1*dsin(a2*(1.d0+eta(i))))
----------------- skr, ckr are sin, cos of k*r-carat
```

```
      skr = dsin(ak * rc)
      ckr = dsqrt(1.d0 - skr*skr)
------------------ multiply by weights now to avoid conversion later
      skr = skr * w(i)/ak
      ckr = ckr * w(i)/ak
------------------ Add intermdiates to the totals, GA..GC
0      iointe = iointe + dcmplx(skr,ckr)
--------------------------- CORRECT FOR JACOBIAN and constant term
       iointe = iointe * dcmplx(a2,0.d0) - dcmplx(0.d0,alpha/ak)
     RETURN
     END
```

APPENDIX B: Subroutines INTE4, SAINT4, INTE3, and SAINT3

```
    subroutine inte4(xp,npe,x,y,z,id,g,h,ak)
-------------------------------------------------------
 PERFORMS INTEGRALS FOR quadrilateral elements not
 CONTAINING THE root node
-------------------------------------------------------
 Inputs:
    XP - ARRAY OF LENGTH 3, CONTAINING THE COORDINATES OF THE NODE
         IN QUESTION (XP(I) = X(I)TH COORDINATE OF THE NODE)
   npe - number of Nodes Per Element
 x,y,z - nodal coordinates of element
    id - element connectivity (used in some shape function routines
    ak - constant from helmholtz equation
 iroot - number of the node in the element which is the root node.
         If the root is outside the element, iroot = 0
-------------------------------------------------------------------
 Outputs:
     g - complex, d.p. array of length npe containing integrals
         OF U* OVER THE ELEMENT
     H - complex, D.P. array of length NPE containing integrals
         OF Q* OVER THE ELEMENT
-------------------------------------------------------------------
  NOTES:
    1. THIS ROUTINE IS CAPABLE OF HANDLING any type of quadrilateral
       element for which a shape function routine is installed
    2. THE INTEGRATION POINTS GIVEN IN /POINT/ ARE FOR INTEGRATION
       OVER line. The integration points must be set before the
       routine is called.  Integration is done in both coordinate
       directions in the usual manner.
    3. Maximum number of integration points presently is 16X16
    4. This routine is valid only for sub-parametric elements
-------------------------------------------------------------------
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION XP(3),X(4),y(4),z(4),id(npe)
    complex*16 g(npe),h(npe),ustar,qstar
-------------- block of 'space' for all of the shape function routines
    common /sspace/ sha(12),xx(3),yy(3),zz(3),u(3),xc(3)
-------------- space for integration points (1-D, gauss type)
    COMMON /QPOINT/ psi(16),W(16),NINPT
-------------- initialize all integrals to zero
    do 10 i = 1,npe
       g(i) = (0.d0,0.d0)
       h(i) = (0.d0,0.d0)
-------------- Find the normal vector and area for the element
-------------- First triangle (nodes 1, 2 & 3)
    call normal(x,y,z,u,al)
-------------- second triangle
    xx(1) = x(2)
    xx(2) = x(3)
    xx(3) = x(4)
    yy(1) = y(2)
```

```
      yy(2) = y(3)
      yy(3) = y(4)
      zz(1) = z(2)
      zz(2) = z(3)
      zz(3) = z(4)
      call normal(xx,yy,zz,u,a2)
      area = a1 + a2
 ------------- INTEGRATE!
      DO 100 I = 1,ninpt
         chi = psi(I)
          c1 = 1.d0 - chi
          c2 = 1.d0 + chi
      do 100 j = 1,ninpt
         eta = psi(j)
        wate = w(i) * w(j)
          e1 = 1.d0 - eta
          e2 = 1.d0 + eta
 ------------- compute geometric SHAPE FUNCTIONS
          sha(1) = 0.25d0 * c1 * e1
          sha(2) = 0.25d0 * c2 * e1
          sha(3) = 0.25d0 * c2 * e2
          sha(4) = 0.25d0 * c1 * e2
 ------------ compute vector from root to integration point, (xc)
        xc(1) = - xp(1)
        xc(2) = - xp(2)
        xc(3) = - xp(3)
        do 50 k = 1,4
           xc(1) = xc(1) + sha(k) * x(k)
           xc(2) = xc(2) + sha(k) * y(k)
)          xc(3) = xc(3) + sha(k) * z(k)
        R  = dsqrt(xc(1)**2 + xc(2)**2 + xc(3)**2)
 ------------ compute interpolation shape functions
        if(npe .ne. 4) call shaqua(chi,eta,npe,id,sha)
 ------------ dot normal, u, and vector, r, to get angle
        cosa = (xc(1)*u(1) + xc(2)*u(2) + xc(3)*u(3))/r
        ustar = dcmplx(dcos(ak * r)*wate/r,-dsin(ak*r)*wate/r)
        qstar = ustar * dcmplx(-cosa/r,-ak*cosa)
 ------------- DO ADDITIONS FOR INTEGRATIONS
        DO 100 k = 1,npe
          H(k) = H(k) + dcmplx(SHA(k),0.d0) * QSTAR
          G(k) = G(k) + dcmplx(SHA(k),0.d0) * USTAR
)0      continue
 ------------- CORRECT FOR JACOBIAN (make temporary use of ustar)
          ustar = dcmplx(area*0.25d0,0.d0)
        DO 110 I = 1,npe
          G(I) = G(I) * ustar
10        H(I) = H(I) * ustar

      RETURN
      END
$EJECT
      SUBROUTINE saint4(npe,x,y,z,id,g,h,ak,iroot)
 ------------------------------------------------------
```

```
PERFORMS INTEGRALS FOR quadrilateral elements
CONTAINING THE root NODE
-----------------------------------------------------
 Inputs:
   npe - number of Nodes Per Element
 x,y,z - nodal coordinates of element
    id - element connectivity (used in some shape function routines
    ak - constant from helmholtz equation
 iroot - number of the node in the element which is the root node.
         If the root is outside the element, iroot = 0
-------------------------------------------------------------------
 Outputs:
     g - complex, d.p. array of length npe containing integrals
         OF U* OVER THE ELEMENT
     H - complex, D.P. array of length NPE containing integrals
         OF Q* OVER THE ELEMENT
-------------------------------------------------------------------
  NOTES:
    1. THIS ROUTINE IS CAPABLE OF HANDLING any type of subparametric
       quadrilateral elements which a shape function routine is
       installed
    2. THE INTEGRATION POINTS GIVEN IN /POINT/ ARE FOR INTEGRATION
       on a line.  These are used in both directions for integrating
       over the element. (Must be set before routine is called)
    3. This routine may call function iointe, which needs 1-D gauss
       points in /spoint/.  These must be set before routine is called
    4. Only valid for planar, subparametric elements
-------------------------------------------------------------------
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION XP(3),X(4),y(4),z(4),id(npe)
    complex*16 g(npe),h(npe),ustar,cointe
-------------- block of 'space' for all of the shape function routines
    common /sspace/ sha(12),xx(3),yy(3),zz(3),u(3),xc(3)
-------------- space for integration points (1-D, gauss type)
    COMMON /QPOINT/ psi(16),W(16),NINPT
-------------- initialize all integrals to zero
    do 10 i = 1,npe
       g(i) = (0.d0,0.d0)
       h(i) = (0.d0,0.d0)
-------------- Area for the element: for first triangle
    call normal(x,y,z,u,a1)
-------------- second triangle
    xx(1) = x(2)
    xx(2) = x(3)
    xx(3) = x(4)
    yy(1) = y(2)
    yy(2) = y(3)
    yy(3) = y(4)
    zz(1) = z(2)
    zz(2) = z(3)
    zz(3) = z(4)
    call normal(xx,yy,zz,u,a2)
    area = a1 + a2
```

```
---------- INTEGRATE!
    DO 100 I = 1,ninpt
       chi = psi(I)
        c1 = 1.d0 - chi
        c2 = 1.d0 + chi
    do 100 j = 1,ninpt
       eta = psi(j)
      wate = w(i) * w(j)
        e1 = 1.d0 - eta
        e2 = 1.d0 + eta
---------- compute geometric SHAPE FUNCTIONS
        sha(1) = 0.25d0 * c1 * e1
        sha(2) = 0.25d0 * c2 * e1
        sha(3) = 0.25d0 * c2 * e2
        sha(4) = 0.25d0 * c1 * e2
---------- compute vector from root to integration point, (xc)
      xc(1) = - xp(1)
      xc(2) = - xp(2)
      xc(3) = - xp(3)
      do 50 k = 1,4
         xc(1) = xc(1) + sha(k) * x(k)
         xc(2) = xc(2) + sha(k) * y(k)
         xc(3) = xc(3) + sha(k) * z(k)
      R  = dsqrt(xc(1)**2 + xc(2)**2 + xc(3)**2)
      ustar = dcmplx(dcos(ak * r)*wate/r,-dsin(ak*r)*wate/r)
---------- Compute interpolation shape functions
      if(npe .ne. 4) call shaqua(chi,eta,npe,id,sha)
---------- DO ADDITIONS FOR INTEGRATIONS
      DO 100 JJ = 1,npe
---------- skip the root node, if one exists
        if(jj .ne. iroot) then
           G(JJ) = G(JJ) + dcmplx(SHA(JJ),0.d0) * USTAR
        end if
0     continue
---------- CORRECT FOR JACOBIAN (make temporary use of ustar)
        ustar = dcmplx(area * 0.25d0,0.d0)
      DO 110 I = 1,npe
.0      G(I) = G(I) * ustar
---------- if root is in the element correct for the constant term
      if(iroot .ne. 0) then
         korn = 4
         g(iroot) = cointe(korn,iroot,x,y,z,ak)
         do 200 i = 1,npe
            if(i .ne. iroot) g(iroot) = g(iroot) - g(i)
0        continue
      end if

    RETURN
    END
EJECT
    SUBROUTINE INTE3(XP,npe,X,y,z,id,G,H,ak)
-------------------------------------------------------
 PERFORMS INTEGRALS FOR subparametric triangular elements not
```

```
CONTAINING THE root node
-----------------------------------------------------------
Inputs:
   XP - ARRAY OF LENGTH 3, CONTAINING THE COORDINATES OF THE NODE
        IN QUESTION (XP(I) = X(I)TH COORDINATE OF THE NODE)
  npe - number of Nodes Per Element
x,y,z - nodal coordinates of element
   id - element connectivity (used in some shape function routines
   ak - constant from helmholtz equation
-----------------------------------------------------------------
Outputs:
    g - complex, d.p. array of length npe containing integrals
        OF U* OVER THE ELEMENT
    H - complex, D.P. array of length NPE containing integrals
        OF Q* OVER THE ELEMENT
-----------------------------------------------------------------
 NOTES:
   1. THIS ROUTINE IS CAPABLE OF HANDLING any type of subparametric
      triangular element
   2. THE INTEGRATION POINTS GIVEN IN /POINT/ ARE FOR INTEGRATION
      for a triangular region. The integration points must be set
      before the routine is called.
   3. Maximum number of integration points presently is 16
   4. This routine is valid only for sub-parametric elements
-----------------------------------------------------------------
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION XP(3),X(3),y(3),z(3),id(npe)
    complex*16 g(npe),h(npe),ustar,qstar
-------------- block of 'space' for all of the shape function routines
    common /sspace/ sha(12),xx(3),yy(3),zz(3),u(3),xc(3)
-------------- space for inte cation points (1-D, gauss type)
    COMMON /TPOINT/ all(16),al2(16),W(16),NINPT
-------------- initialize all integrals to zero
    do 10 i = 1,npe
       g(i) = (0.d0,0.d0)
       h(i) = (0.d0,0.d0)
-------------- Find the normal vector and area for the element
    call normal(x,y,z,u,area)
-------------- INTEGRATE!
    DO 100 I = 1,ninpt
-------------- use shape functions to caluculate vector from root
               to point in the element, (xc)
      xc(1) = all(i)*(x(1)-x(3)) + al2(i)*(x(2)-x(3)) + x(3) - xp(1)
      xc(2) = all(i)*(y(1)-y(3)) + al2(i)*(y(2)-y(3)) + y(3) - xp(2)
      xc(3) = all(i)*(z(1)-z(3)) + al2(i)*(z(2)-z(3)) + z(3) - xp(3)
      R  = dsqrt(xc(1)**2 + xc(2)**2 + xc(3)**2)
------------ compute interpolation shape functions
      call shatri(all(i),al2(i),npe,id,sha)
------------ dot normal, u, and vector, r, to get angle
      cosa = (xc(1)*u(1) + xc(2)*u(2) + xc(3)*u(3))/r
      ustar = dcmplx(dcos(ak * r)*w(i)/r,-dsin(ak*r)*w(i)/r)
      qstar = ustar * dcmplx(-cosa/r,-ak*cosa)
------------ DO ADDITIONS FOR INTEGRATIONS
```

```
      DO 100 k = 1,npe
        H(k) = H(k) + dcmplx(SHA(k),0.d0) * QSTAR
        G(k) = G(k) + dcmplx(SHA(k),0.d0) * USTAR
0     continue
------------- CORRECT FOR JACOBIAN (make temporary use of ustar)
        ustar = dcmplx(area,0.d0)
      DO 110 I = 1,npe
        G(I) = G(I) * ustar
.0      H(I) = H(I) * ustar

    RETURN
    END
    SUBROUTINE saint3(npe,x,y,z,id,g,h,ak,iroot)
-----------------------------------------------------
 PERFORMS INTEGRALS FOR subparametric triangular elements not
 CONTAINING THE root node
-----------------------------------------------------
 Inputs:
   npe - number of Nodes Per Element
 x,y,z - nodal coordinates of element
    id - element connectivity (used in some shape function routines
    ak - constant from helmholtz equation
 iroot - local number of the root node
-------------------------------------------------------------
 Outputs:
     g - complex, d.p. array of length npe containing integrals
         OF U* OVER THE ELEMENT
     H - complex, D.P. array of length NPE containing integrals
         OF Q* OVER THE ELEMENT
-------------------------------------------------------------
  NOTES:
    1. THIS ROUTINE IS CAPABLE OF HANDLING any type of subparametric
       triangular element
    2. THE INTEGRATION POINTS GIVEN IN /POINT/ ARE FOR INTEGRATION
       for a triangular region. The integration points must be set
       before the routine is called.
    3. Maximum number of integration points presently is 16
    4. This routine is valid only for sub-parametric elements
-------------------------------------------------------------
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION XP(3),X(3),y(3),z(3),id(npe)
    complex*16 g(npe),h(npe),ustar,cointe
------------- block of 'space' for all of the shape function routines
    common /sspace/ sha(12),xx(3),yy(3),zz(3),u(3),xc(3)
------------- space for integration points (1-D, gauss type)
    COMMON /TPOINT/ all(16),al2(16),W(16),NINPT
------------- initialize all integrals to zero
    do 10 i = 1,npe
       g(i) = (0.d0,0.d0)
)      h(i) = (0.d0,0.d0)
------------- Find the normal vector and area for the element
    call normal(x,y,z,u,area)
------------- INTEGRATE!
```

```
    DO 100 I = 1,ninpt
---------- use shape functions to caluculate vector from root
           to point in the element, (xc)
      xc(1) = all(i)*(x(1)-x(3)) + al2(i)*(x(2)-x(3)) + x(3) - xp(1)
      xc(2) = all(i)*(y(1)-y(3)) + al2(i)*(y(2)-y(3)) + y(3) - xp(2)
      xc(3) = all(i)*(z(1)-z(3)) + al2(i)*(z(2)-z(3)) + z(3) - xp(3)
      R  = dsqrt(xc(1)**2 + xc(2)**2 + xc(3)**2)
---------- compute interpolation shape functions
      call shatri(all(i),al2(i),npe,id,sha)
---------- dot normal, u, and vector, r, to get angle
      ustar = dcmplx(dcos(ak * r)*w(i)/r,-dsin(ak*r)*w(i)/r)
---------- DO ADDITIONS FOR INTEGRATIONS
      DO 100 k = 1,npe
        if(k .ne. iroot) G(k) = G(k) + dcmplx(SHA(k),0.d0) * USTAR
0     continue
---------- CORRECT FOR JACOBIAN (make temporary use of ustar)
        ustar = dcmplx(area,0.d0)
      DO 110 I = 1,npe
0       G(I) = G(I) * ustar
---------- correct for root node
        g(iroot) = cointe(npe,x,y,z,ak)
      do 120 i = 1,npe
0        if(i .ne. iroot) g(iroot) = g(iroot) - g(i)

    RETURN
    END
```

PENDIX C: Subroutines NORMAL, SHAQUA SHZDQQ, SHATRI, and SH2DCQ

```
=================================================================
    subroutine normal(x,y,z,u,area)
-----------------------------------------------------------------
 Computes the normal and area of a triangular plane in three
    dimensions
-----------------------------------------------------------------
 Inputs:
    x,y,z - coordinates of the three nodes
        u - normalized normal vector formed by crossing the vector
            between nodes 1 and 2 into the vector between nodes
            1 and 3
     area - area of the triangle
---------------- COMPUTE VECTORS R1 & R2 ALONG SIDE 1 & 3 OF ELEMENT
    implicit real*8 (a-h,o-z)
    dimension x(3),y(3),z(3),u(3),r1(3),r2(3)
-------------- form the two vectors to be crossed
    r1(1) = x(2) - x(1)
    r1(2) = y(2) - y(1)
    r1(3) = z(2) - z(1)
    r2(1) = x(3) - x(1)
    r2(2) = y(3) - y(1)
    r2(3) = z(3) - z(1)
-------------- CROSS R1 & R2 TO FIND U, NORMAL TO THE SURFACE
    U(1) = (R1(2) * R2(3) - R1(3) * R2(2))
    U(2) = (R1(3) * R2(1) - R1(1) * R2(3))
    U(3) = (R1(1) * R2(2) - R1(2) * R2(1))
-------------- THE MAGNITUDE OF U IS TWICE THE AREA OF ELEMENT
    BOT = DSQRT(U(1)**2 + U(2)**2 + U(3)**2)
    AREA = 0.5 * BOT
-------------- NORMALIZE U
    U(1) = U(1)/BOT
    U(2) = U(2)/BOT
    U(3) = U(3)/BOT

    return
    end
=================================================================
*****************************************************************
  MODULE FOR 2-D planar shape functions
*****************************************************************

    SUBROUTINE SHaqua(SS,TT,npe,id,SHA)
-----------------------------------------------------------------
 CONTROL FOR TWO-DIMENSIONAL ISOPARMETRIC ELEMENTS
 TYPES SUPPORTED:
  NUMBER      HIERARCHIAL              TYPE OF ELEMENT
OF NODES      NODES POSSIBLE
--------      -------------            ---------------------------
   4             ---                   LINEAR QUADRILATERAL
   8             YES                   QUADRATIC QUADRILATERAL
```

```
  9             YES                     LAGRANGE QUADRILATERAL
 12             YES                     CUBIC QUADRILATERAL
C-----------------------------------------------------------------
 INPUT ARGUMENTS:
SS,TT  -- NATURAL COORDINATES
  npe  -- number of nodes in element
   ID  -- element connectivity
C-----------------------------------------------------------------
Outputs:
  SHA  -- SHAPE FUNCTIONS AT (SS,TT)
C-----------------------------------------------------------------
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION S(4),T(4),SHA(npe)
    DATA S/-0.5D0,0.5D0,0.5D0,-0.5D0/,T/-0.5D0,-0.5D0,0.5D0,0.5D0/
C-----  FORM 4-NODE QUADRILATERAL SHAPE FUNCTIONS
    DO 100 I=1,4
)       SHA(I)= (0.5+S(I)*SS)*(0.5 + T(I)*TT)
C----- ADD higher order terms if necessary
     IF(NPE .EQ. 8 .or. npe .eq. 9) CALL SH2DQQ(SS,TT,SHA,ID,NPE)
     IF(NPE .EQ. 12) CALL SH2DCQ(SS,TT,SHA,ID,NPE)

    RETURN
    END
EJECT
C-----------------------------------------------------------------

    SUBROUTINE SH2DQQ(S,T,SHA,IX,NPE)
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION IX(npe),SHA(npe)
    S2 = 0.5d0 * (1.d0 - S*S)
    T2=  0.5d0 * (1.d0 - T*T)
        DO 100 I=5,npe
0       SHA(I) = 0.
C---------------------------------------- MIDSIDE NODES
    if(ix(5) .ne. 0) SHA(5) =  S2 * (1.d0 - T)
    if(ix(6) .ne. 0) SHA(6) =  T2 * (1.d0 + S)
    if(ix(7) .ne. 0) SHA(7) =  S2 * (1.d0 + T)
    if(ix(8) .ne. 0) SHA(8) =  T2 * (1.d0 - S)
C---------------------------------------- LAGRANGE HEREAFTER
    if(npe .eq. 9) then
    if(ix(9) .ne. 0) then
      SHA(9) =  4. * S2 * T2
C---------------------------------------- CORRECT EDGE FOR INTERIOR
      DO 105 I=1,4
        SHA(I) = SHA(I) - 0.25 * SHA(9)
5       IF(IX(I+4) .NE. 0) SHA(I+4) = SHA(I+4) - .5 * SHA(9)
    end if
    end if
C---------------------------------- CORRECT CORNER FOR MIDSIDE
7   K=8
        DO 109 I=1,4
        L=I+4
        SHA(I) = SHA(I) - .5*(SHA(K) + SHA(L))
```

```
9       K=L
      RETURN
      END
EJECT
      SUBROUTINE SHATRI(A1,A2,NPE,IX,SHA)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 L(3)
      DIMENSION SHA(npe),S2(3),S3(3),IX(npe)
      DATA S2/1.D0,0.D0,-1.D0/,S3/0.D0,1.D0,-1.D0/
------------------------------ FORM THIRD SHAPE FUNCTION
      L(1) = A1
      L(2) = A2
      L(3) = 1. - L(1) - L(2)
-------------------------------- FORM LINEAR SHAPE FUNCTIONS
      DO 10 I=1,3
          SHA(I) = L(I)

      IF(NPE .NE. 6) RETURN

          DO 20 J=4,6
0     SHA(J) = 0.d0
------------------------------ FORM QUADRATIC TERMS AS NECESSARY
      if(ix(4) .ne. 0) SHA(4) = 4.d0 * L(1) * L(2)
      if(ix(5) .ne. 0) SHA(5) = 4.d0 * L(2) * L(3)
      if(ix(6) .ne. 0) SHA(6) = 4.d0 * L(3) * L(1)
------------------------------ correct corners for midside nodes
      KK = 6
      DO 60 I=1,3
          K=I+3
      SHA(I) = SHA(I) - 0.5d0*(SHA(KK) + SHA(K))
1     KK=K

      RETURN
      END
EJECT
      SUBROUTINE SH2DCQ(S,T,SHA,IX,NPE)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION IX(npe),SHA(npe)
                1     2     3     4     5
      DATA S1/0.D0,0.D0,0.D0,0.D0,-1.D0
          6     7     8     9     10    11    12
     * ,1.D0,1.D0,1.D0,1.D0,-1.D0,-1.D0,-1.D0/
                1     2     3     4     5     6
      DATA T1/0.D0,0.D0,0.D0,0.D0,-1.D0,-1.D0,
          7     8     9     10    11    12
     * -1.D0,1.D0,1.D0,1.D0,1.D0,-1.D0/

      C  = 9.d0/32.d0
      SS = 1.d0 - S*S
      TT = 1.d0 - T*T
      S2 = 0.5d0 * SS
      T2 = 0.5d0 * TT
      DO 10 I=5,12
```

```
      SHA(I)=0.d0
------------------------- ALLOW FOR MISSING NODES
      IF(IX(5) .ne. 0) then
      IF(IX(6) .ne. 0) then
          SHA(5) = C*(1.d0 - T) * SS * (1.d0 - 3.d0*S)
          SHA(6) = C*(1.d0 - T) * SS * (1.d0 + 3.d0*S)
          SHA(1) = SHA(1) - (2.d0 * SHA(5) + SHA(6))/3.d0
          SHA(2) = SHA(2) - (2.d0 * SHA(6) + SHA(5))/3.d0
      else
          SHA(5) = S2 * (1.d0 - T)
          SHA(1) = SHA(1) - 0.5d0 * SHA(5)
          SHA(2) = SHA(2) - 0.5d0 * SHA(5)
      end if
      end if
-------------------- side 2, nodes 7 and 8
      IF(IX(7) .ne. 0) then
      IF(IX(8) .ne. 0) then
          SHA(7) = C*(1.d0 + S) * TT * (1.d0 - 3.d0*T)
          SHA(8) = C*(1.d0 + S) * TT * (1.d0 + 3.d0*T)
          SHA(2) = SHA(2) - (2.d0 * SHA(7) + SHA(8))/3.d0
          SHA(3) = SHA(3) - (2.d0 * SHA(8) + SHA(7))/3.d0
      else
          SHA(7) =  T2 * (1.d0 + S)
          SHA(2) = SHA(2) - 0.5d0 * SHA(7)
          SHA(3) = SHA(3) - 0.5d0 * SHA(7)
      end if
      end if
------------------- side 3, nodes 9 and 10
      IF(IX(9 ) .ne. 0) then
      IF(IX(10) .ne. 0) then
          SHA(9)  = C*(1.d0 + T) * SS * (1.d0 + 3.d0 * S)
          SHA(10) = C*(1.d0 + T) * SS * (1.d0 - 3.d0 * S)
          SHA(3) = SHA(3) - (2.d0 * SHA(9 ) + SHA(10))/3.d0
          SHA(4) = SHA(4) - (2.d0 * SHA(10) + SHA(9 ))/3.d0
      else
          SHA(9) = S2 * (1.d0 + T)
          SHA(3) = SHA(3) - 0.5d0 * SHA(9)
          SHA(4) = SHA(4) - 0.5d0 * SHA(9)
      end if
      end if
------------------ side 4, nodes 11 and 12
      IF(IX(11) .ne. 0) then
      IF(IX(12) .ne. 0) then
          SHA(11) = C*(1.d0 - S) * TT * (1.d0 + 3.d0 * T)
          SHA(12) = C*(1.d0 - S) * TT * (1.d0 - 3.d0 * T)
          SHA(4) = SHA(4) - (2.d0 * SHA(11) + SHA(12))/3.d0
          SHA(1) = SHA(1) - (2.d0 * SHA(12) + SHA(11))/3.d0
      else
          SHA(11) = T2 * (1.d0 - S)
          SHA(4) = SHA(4) - 0.5d0 * SHA(11)
          SHA(1) = SHA(1) - 0.5d0 * SHA(11)
      end if
      end if
      return
      END
```